Sindu Bhowmik

Internal Discussion 5/7/20

Summary

Sindu Bhowmik and co-workers have developed a unique way to analyze molecular dynamics trajectories using variational autoencoders. The trajectories are run through a CNN-based variational autoencoder to compute a latent representation of each configuration of the simulation. The latent space representations are then clustered to understand the structure and dynamics of the proteins. The third phase of the workflow involving AI-enabled docking is not ready and we did not discuss further.

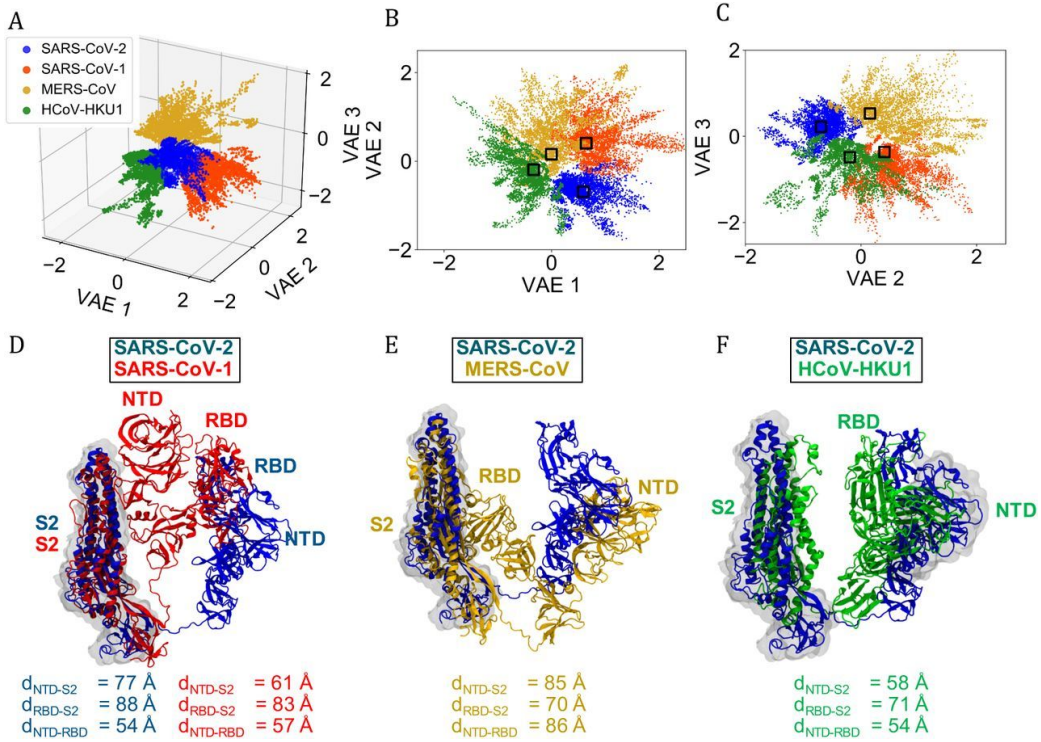
VAE Used to Analyze MD Simulations

MD yields **x**(t), **y**(t), **z**(t) for t={0,...,20000}, **x**=[x_0,...,x_{1200}]

For each t, take **x**,**y**,**z** and construct distance matrix, **D**.

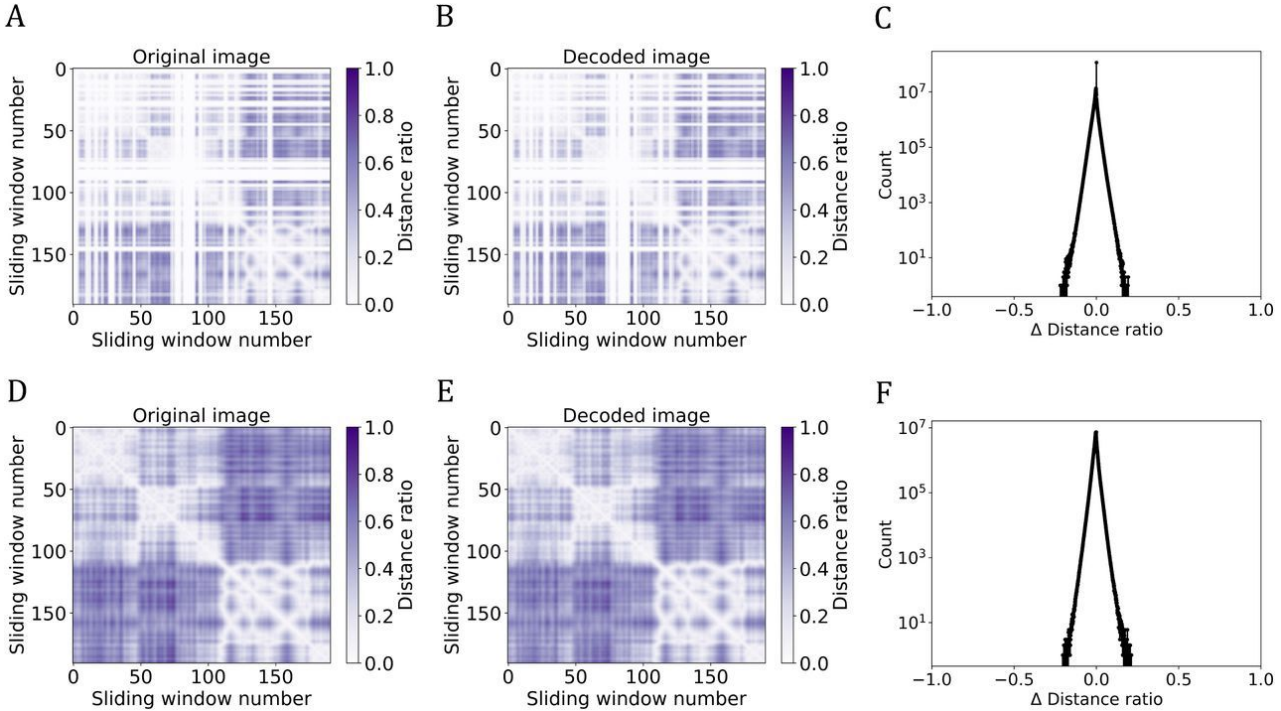
Train autoencoder on **D**.

VAE yields an embedding of each MD configuration



Distance Matrix

reduce D from 1200x1200 to around 400x400 by hand.



Encoder

```
(encoder): Sequential(
        (0): Conv2d(1, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): SELU(inplace=True)
        (2): LayerNorm((8, 1068, 1068), eps=1e-05, elementwise_affine=True)
        (3): Conv2d(8, 16, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
        (4): SELU(inplace=True)
        (5): LayerNorm((16, 534, 534), eps=1e-05, elementwise_affine=True)
        (6): Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
        (7): SELU(inplace=True)
        (8): LayerNorm((32, 267, 267), eps=1e-05, elementwise_affine=True)
        (9): Flatten()
        (10): Linear(in_features=2281248, out_features=100, bias=True)
)
```

Decoder

```
(decoder): Sequential(
        (0): Linear(in_features=50, out_features=2281248, bias=True)
        (1): SELU(inplace=True)
        (2): Reshape(sizes=(32, 267, 267), batch_dims=1)
        (3): LayerNorm((32, 267, 267), eps=1e-05, elementwise_affine=True)
        (4): ConvTranspose2d(32, 16, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
        (5): SELU(inplace=True)
        (6): LayerNorm((16, 534, 534), eps=1e-05, elementwise_affine=True)
        (7): ConvTranspose2d(16, 8, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
        (8): SELU(inplace=True)
        (9): LayerNorm((8, 1068, 1068), eps=1e-05, elementwise_affine=True)
        (10): ConvTranspose2d(8, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (11): Sigmoid()
 )
```

Model Size

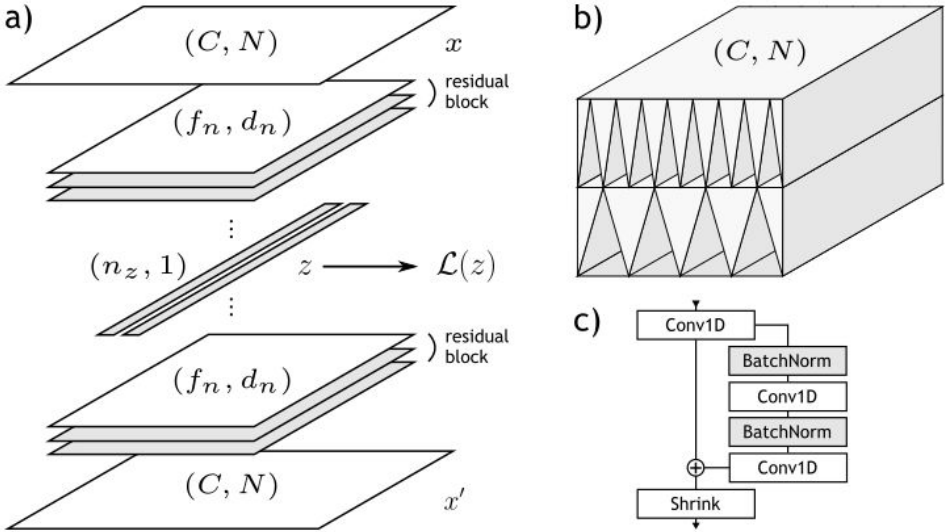
> encoder.0.weight 72
> encoder.0.bias 8
> encoder.2.weight 9,124,992
> encoder.2.bias 9,124,992
> encoder.3.weight 1,152
> encoder.3.bias 16
> encoder.5.weight 4,562,496
> encoder.5.bias 4,562,496
> encoder.6.weight 4,608
> encoder.6.bias 32
> encoder.8.weight 2,281,248
> encoder.8.bias 2,281,248
> encoder.10.weight 228,124,800
> encoder.10.bias 100

> decoder.0.weight 114,062,400
> decoder.0.bias 2,281,248
> decoder.3.weight 2,281,248
> decoder.3.bias 2,281,248
> decoder.4.weight 4,608
> decoder.4.bias 16
> decoder.6.weight 4,562,496
> decoder.6.bias 4,562,496
> decoder.7.weight 1,152
> decoder.7.bias 8
> decoder.9.weight 9,124,992
> decoder.9.bias 9,124,992
> decoder.10.weight 72
> decoder.10.bias 1

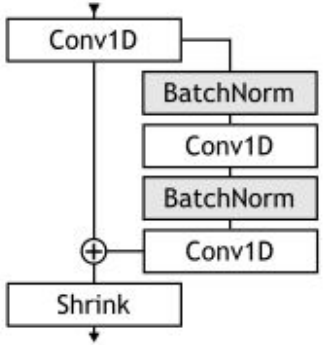
Fully Convolutional VAE

- 1D convolutions in a residual block configuration
- 11 residual-conv blocks in encoder
- 45 conv layers total in encoder
- 111 M parameters
- 64 batch size

Note: number of samples must be an exact multiple of batch size



Fully Convolutional VAE




```
def Residual1DConv(x,
                   filters,
                   kernel_size,
                   activation='relu',
                   name='res1',
                   shrink=False,
                   kfac=2):

    res = Conv1D(filters,
                 kernel_size=1,
                 padding='same',
                 name=name+'_1x1')(x)

    x = BatchNormalization(name=name+'_bn1')(x)
    x = Activation(activation, name=name+'_act1')(x)
    x = Conv1D(filters,
                 kernel_size,
                 padding='same',
                 name=name+'_conv1D1')(x)

    x = BatchNormalization(name=name+'_bn2')(x)
    x = Activation(activation, name=name+'_act2')(x)
    x = Conv1D(filters,
                 kernel_size,
                 padding='same',
                 name=name+'_conv1D2')(x)
    x = Add(name=name+'_add')([x, res])

    if shrink:
        x = Conv1D(filters=filters,
                 kernel_size=kfac,
                 strides=kfac,
                 padding='same',
                 activation=activation,
                 name=name+'strided')(x)

    return x
```

Model Definition: Encoder Res Blocks

- Residual blocks are defined in a loop
- Note how the number of filters is computed

```
for lidx in range(params.enc_reslayers):

    filters = params.enc_filters * params.enc_filter_growth_fac**lidx
    filters = int(round(filters))

    x = Residual1DConv(x,
           filters,
           params.enc_kernel_size,
           name='res'+str(lidx),
           shrink=True)
```

Instantiation

- Let’s look at the Jupyter Notebook to see an instantiation

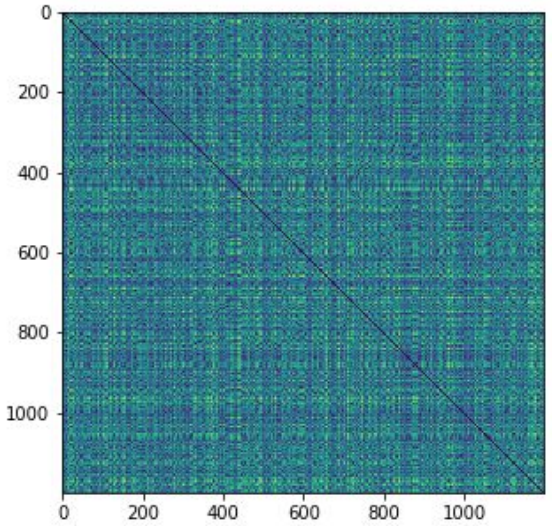
```
encoder.summary()
```

```
Model: "encoder"
_________________________________________________________________
Layer (type)                 Output Shape
=================================================================
input_smiles (InputLayer)    (None, 1200, 1200)
```

Preparing Synthetic Data

```
n_steps = 50      # similar to number of MD configurations
n_atoms = 1200    # atoms
n_coords = 3      # 3D: x, y, z
hidden_dim = 150  # latent space dim
batch_size = 10

xyz = np.random.random((n_steps, n_atoms, n_coords))
dismat = np.array([ distance_matrix(i, i) for i in xyz ])
y_true = np.zeros((n_steps, hidden_dim*2))
dismat.shape
```



Command Line Arguments

- For help: ./run_fcn_vae.py -h
- Shown here is how to run inside a Jupyter notebook
- Let's do a quick single GPU run

```
p = fcn_parameters()
params = p.parse_args(['--NCHARS=1200',
                       '--MAX-LEN=1200',
                       '--hidden-dim=%d'%hidden_dim,
                       '--dec-filters=1200',
                       '--dec-reslayers=3',
                       '--batch-size=%d'%batch_size,
                       '--optimizer=adam'
                      ])
params
```

Running Multi-GPU

- Data parallel implemented with Horovod
- OpenMPI provided inside the container
- Kick off with mpirun

```
mpirun -np 2 python run_fcn_vae.py \
    --batch-size 32 \
    --random-train-size 128 \
    --random-train-data
```

```
Initialized synthetic data shaped:  (128, 1200, 1200)
Initializing Horovod rank 0
Initializing Horovod rank 1
```

Environment

- requirements.txt provided. Build a conda env:

```
conda create -n tf -f requirements.txt
```

- Dockerfile included. Build container as:

```
docker build -f Dockerfile -t tf_vae .
```

- Makefile included. Edit Makefile, set variables, run as:

```
make run
```